Software COTS Components – Problems, And Solutions?

Ted Dowling
Systems & Software Engineering Centre
Defence Evaluation & Research Agency (DERA)
St Andrew's Road
Malvern
WR14 3PS
UK
+44 (0) 1684 894734
ejdowling@sec.dera.gov.uk

Summary. COTS components offer a solution to many obsolescence problems, but certain COTS items can also introduce their own difficulties. Commercial operating systems, for example, play a key system role but are single-source and black box, denying the user both the visibility and control of a bespoke item. Open source software in general, but the Linux operating system in particular, seems to offer many of the advantages of COTS but with the added benefit of full access to the source code. However, the widespread adoption of Linux presents not only opportunities but some potential difficulties, for which a possible solution is a dedicated focus within the defence community.

Background to Paper. The Defence Evaluation and Research Agency (DERA) is the prime source of research for the UK Ministry of Defence (MOD), and also provides a major source of independent advice to MOD during all stages of systems procurement and deployment.

The Systems and Software Engineering Centre (SEC) is a relatively new body within DERA, being established in 1994 to act as a focus for professional software (and soon after, systems) engineering within DERA. The majority of its complement of about 260 staff have an industrial background.

The SEC has responsibility for the systems and software standards and practices used across DERA (which has a staff of around 11,000). It provides the editor for the draft ISO standard (ISO15288) on systems engineering and is influential in setting the systems engineering direction of MOD's procurement arm, the Defence Procurement Agency (DPA). It provides systems and software engineering support to a wide range of programmes within DPA, and increasingly to the Defence Logistics Organisation of MOD. The SEC is also leading in the field of capability assessment and evaluation, eg in developing and applying various Capability Maturity Models (CMMs). The author is the SEC's Technical Manager.

Despite this background, it should be made clear that this paper does not constitute the results from a MOD-funded research programme, nor does it represent the official view of MOD, DERA or the SEC. Rather it captures the personal views and thinking of the author. However, the author is pleased to acknowledge the rich source of ideas he has encountered in the SEC, DERA, MOD, Defence Scientific Advisory Council (DSAC) working parties and other contexts.

Some of the ideas addressed here are included in a recent paper in the Journal of Defence Science, published by DERA.

Introduction. Commercial components[1] that can be used in defence systems may take several forms:

- A common commercial item from the civil world (eg a PC or Land Rover)
- A specialised item from the civil world (eg an inertial navigation system box for an aircraft)
- A "standard" item from the defence world (eg a gun sight)
- An item that has been used before but just needs "a little change" to make it meet its new purpose – especially true of software
- A small component (eg a processor chip)
- A full service (eg satellite communication)

Whichever kind of component is considered, there are some common issues that arise when thinking about a defence system in which commercial components play a significant part[2]. These issues are highlighted most starkly when the item is a truly commercial (COTS) system component. Hence, this is the case considered in this paper.

[1] In this paper, the term "component" is generally used broadly and is intended to encompass sub-systems; it is not intended to imply the lowest level in a decomposition.

[2] While acquisition of a capability might be more appropriate to consider than acquisition of a system, systems are considered for convenience.

Paper presented at the RTO SCI Symposium on "Strategies to Mitigate Obsolescence in Defense Systems Using Commercial Components", held in Budapest, Hungary, 23-25 October 2000, and published in RTO MP-072.

While many of the issues discussed apply equally to any kind of COTS, the emphasis of this paper is on software items. There are a number of reasons for this:

- Software is the difficult and expensive part of most systems
- Software COTS items are often the most sophisticated and complex kind of component
- Many software COTS items change rapidly

COTS and Obsolescence. COTS items have a particular attraction when considering obsolescence management. They typically have a longer lifetime than bespoke components because they have a much broader customer base (or at least are cheaper over a given lifetime because maintaining bespoke items is expensive). There is also often an opportunity for multi-sourcing that is very significant.

More generally, compared with a bespoke approach, a COTS-based development often offers many advantages, including:

- Reduced costs
- Reduced timescales
- Increased reliability through exploitation of proven items
- Exploitation of civil research and development (R&D) investment, which globally has far outstripped defence-focused R&D
- Accelerated introduction through familiar user interfaces that facilitate training, etc
- Increased opportunity for multi-sourcing through open standards
- Improved interoperability between defence systems and organisations, through standardisation
- Improved interoperability between defence and non-defence systems and organisations, again through standardisation

It is important to note that a COTS-based solution will not necessarily offer all these advantages. For example, the COTS item might be proprietary and single-source, while still delivering all the other advantages listed above.

Impact of COTS. The potential benefits are thus great, but it is vital to consider carefully how the use of COTS impacts some key areas:

- Military capability
- Systems development
- System acquisition and support

Capability. COTS is a powerful influence towards a "level playing field". By its very nature, a COTS component must be considered to be available to any country and organisation. We must assume that potential enemies can:

- Exploit the same COTS components in their own systems
- Infer how we might use them in our systems
- Explore their weaknesses
- Develop countermeasures of various kinds
- Perhaps exploit upgrades to the COTS components faster than we can

Because of their ubiquitous nature, COTS items are a particularly vulnerable part of a system. Where the item is something like an operating system, we can expect any flaw such as a security weakness to be identified very publicly. Perhaps more dangerous is the community of semi-underground "crackers" - individuals, and sometimes organisations, who spend time identifying weaknesses in COTS software items and then publish or exchange this information on the Internet.

Conversely, it is possible to exploit this published information and counter any weaknesses rapidly - assuming the underlying mechanisms for rapid upgrading are in place, a crucial point. Of course, if the affected item is a COTS package over which we have no direct control, the best we can do with the information is to press the supplier for a fix and attempt a "work around" until it arrives.

System Development. A key characteristic of most COTS components is that they are "black boxes". This has a number of serious implications:

- We cannot be sure what they contain; they may have in them – perhaps deliberately – features that compromise or destroy aspects such as security
- We cannot examine how they achieve their functionality; we cannot, for example, apply techniques such as static code analysis when assessing their role in a safety-critical context

In practice, some component suppliers may be willing to provide internal details, although especially where national boundaries are crossed, this may require some negotiation.

System Acquisition and Support. The impact of COTS on acquisition can be seen most clearly in Table 1 that compares the "traditional" approach, in which the customer (say, NATO) fully specified all system components, to a COTS-based procurement.

TRADITIONAL	COTS-BASED
NATO able to plan and control system development	COTS components change asynchronously and rapidly
NATO able to define functionality	COTS supplier defines functionality to suit larger market. NATO spec may preclude use of COTS if too rigid.
NATO able to control/view development process to support its responsibilities for certification, etc	COTS item is "black box" and alternative approaches to certification, etc may be needed
NATO able to control interfaces and interoperability	Interoperability may be enhanced if same COTS component in both systems, but otherwise may be very difficult because COTS interfaces not fully defined/maintained
NATO able to exploit expertise, standards, etc for component engineering	Key activity now becomes systems integration – more of a "black art"
NATO able to control functionality	COTS supplier may define upgrade package (eg operating system plus applications)
NATO able to co-ordinate change to component with change to whole system	COTS component change driven purely by commercial factors, not synchronised with system constraints (eg refits). May lead to many variants of equipment fit across fleet of platforms.
NATO able to procure changes/fix problems, especially in emergency, perhaps in the field	COTS component changed if and when supplier sees market advantage; NATO not a significant customer
NATO able to assume component will remain available (especially components that wear out)	COTS component may simply cease to be available (not just be unsupported) if commercial market moves away from it

Table 1 Traditional v COTS-Based Acquisition

Fundamental characteristics of COTS items that might be exploited in NATO systems can thus be summarised – perhaps rather starkly – as:

- "Take it or leave it" functionality
- Rapid change
- Out of NATO control

COTS Software. There are thus two key issues that apply to all COTS items but which are especially problematical for COTS software.

The first is *control*. A major advantage of bespoke components is that the customer can control the functionality, interfaces, schedule, upgrade path, etc. Conversely, with COTS items, the customer is not the leader but the led. By its nature, COTS software is subject to much more variation in functionality and interfaces than a relatively constrained item such as a processor chip.

The second issue is *visibility*. Bespoke software can be examined to ensure it does not include any feature to prejudice security, safety, etc. In contrast, most COTS items are "black box". It is generally accepted these days that obscurity is not the same as security and that on balance, the interests of security are best served by transparency. Again, visibility is an issue that is particularly relevant for software since software is generally much more complex and flexible that hardware.

Within the field of COTS software, the operating system (OS) is especially key. It is central to the whole capability implemented in software and can be very complex. It also pivotal in the sense that very often a change to the OS for whatever reason can mean a change to the applications running on it. This is particularly troublesome if, for example, a change of OS is needed to fix a bug and this solution only comes as a package that introduces new problems, in the shape of revised applications.

It is interesting to note that in this way, a change to the COTS OS can make the applications that run on it obsolete. That is, the COTS system element can actually make proprietary system

elements obsolete - a perhaps unexpected situation. Of course, this usually arises where the OS is COTS, but is also itself proprietary and single-source.

Note that in this paper, the emphasis is on general purpose operating systems rather than more specialist examples, such as real-time operating systems for embedded processors.

Addressing the Problems. The capability impact addressed above - eg where a potential enemy has the same capability as ourselves through using the same COTS item - requires that we undertake a much broader review of issues. Major changes in strategy may be driven by answers to questions such as:

- Where does our COTS-based system have the edge over an enemy's system that uses the same COTS items?
- Where are the weaknesses in our system that the COTS items introduce? How might an enemy exploit these? What countermeasures can we put into place?
- If we have to conclude that our COTS-based system does not offer significant, dependable superiority of technology, is there some other source of military advantage for us, such as superiority of training or numbers?

The systems design issues - when we no longer have visibility of the internal behaviour of the COTS item - implies that we have to consider means of containing any undesirable behaviour and preventing it impacting on the rest of the system. Such approaches often take the form of "wrapping" of some kind, but introducing parallel COTS components from different sources and using voting may be an alternative in some circumstances.

Also key for system design is the question of standards. COTS components are usually associated with standards of various kinds. These standards may address the interfaces/ interoperability of the item and/or its functionality. Standards may be *de jure*, typically endorsed by an international standards body or broad-based industry group, or *de facto*, typically set by a single, dominant supplier.

In either case, the choice of standard during development is crucial. Important questions include:

- How stable is the standard?
- How definitive is it? (eg can widely-differing items both claim compliance?)
- How many vendors/products support the standard now?
- How many will support it in the future?
- Is any replacement standard likely to offer upwards compatibility?

Of course, standards are most important from an obsolescence viewpoint if the strategy is to use them to define a "hole" in the system into which can be "plugged" a variety of products from a variety of suppliers, all of which "fit". This is one approach, but in some areas, others are also possible.

The Best of Both Worlds? As noted above, the key issues for "traditional" COTS software are control and visibility. These are the characteristics that have to be traded off against the advantages offered by a COTS item. However, one class of software component does appear to offer the best of the bespoke and COTS worlds: open source software.

Open source software is freely available to all interested parties. It may be copied, changed and distributed onwards. Thus to all intents and purposes, it can be owned in the same way as bespoke software. However, it may not be totally without restriction. For example, usually it *is* licensed and the licence stipulates that if it is changed then the modified version cannot be redistributed unless it too is freely available.

Open source software (OSS) therefore offers the control and visibility of bespoke software while at the same time avoiding the cost and risk of developing the software from scratch.

Furthermore, the OSS items tend to be exploited and modified by a wide community of users across the globe, communicating via the Internet. The OSS item is thus essentially "owned" by a wide community of enthusiastic people who want to see it succeed. Because the source is available to them, they are able to identify problems by analysing the code rather than simply waiting until some erroneous behaviour is spotted. They are also able – and motivated – to devise solutions and disseminate these to others.

This cultural dimension of OSS is a very significant factor in its success.

Linux. The best known example of OSS is undoubtedly Linux. It is important to note that some of the ways Linux has developed and its current position are not necessarily typical of other OSS items, and OSS components do not necessarily have to follow the Linux model. However, Linux is so significant that it deserves attention.

Strictly, Linux is an operating system consisting of the kernel (that provides the basic mechanisms for scheduling, etc) and device drivers that allow

it to communicate with various peripherals. However, in practice, the term is also used to apply to a wider collection of items, including, for example, a graphical user interface.

The strict interpretation is useful to bear in mind since while the kernel is essentially standardised (see below), there are a number of different options – virtually all open source themselves – that exist for the applications that go with the kernel to make it an "operating system" in the Microsoft Windows sense – ie the kernel plus a whole host of other things that actually allow the user to do something useful. Thus the various Linux packages ("distributions") that are available from a variety of suppliers all provide a different set of items together with the standard Linux core.

The Linux kernel is essentially controlled by the originator of Linux, Linus Torvalds. Typically, somebody in the world will identify the need for a new feature and publish a very early and imperfect version of it. Others will use and refine this, also publishing their work. Eventually, the item will become stable and widely used and then accepted into the "official" version by Torvalds.

There are several implications of this, of course. On the positive side, it is very visible in which way the product is moving and interested parties can at the very least monitor this. They can also influence it if they are prepared to actually contribute development effort. The disadvantage is that in fact it may be moving in conflicting directions (eg there are currently a number of different hard real-time extensions being developed) and the final outcome might not be at all clear.

There is also an obvious issue over the role of Torvalds as the arbiter of what constitutes a formal release of Linux. His role is vital in deciding when a new release occurs and what it contains, and as the system grows this becomes ever more demanding. Already, there are some indications that releases are slipping behind schedule (eg version 2.4). There is also some uncertainty about what will happen when Torvalds, for whatever reason, relinquishes this role.

Technically, Linux is a flavour of Unix, although it is worth noting that it does not fully comply with the Open Group's criteria that would allow it to use the Unix trademark. Partly due to the way it has been developed, it is very modular and has a relatively low number of system interfaces when compared to Windows, for example (230 rather than 3500).

A key feature is that Linux has been implemented on a very wide range of machines. This is facilitated by its modularity and relative simplicity, making it possible to run it on "bottom end" architectures.

Other (Open Source) Software. The same basic approach to development that has proved so popular for Linux has been adopted for many other pieces of software, although a lot have followed a more traditional, and some would say more controlled, development process. Not surprisingly, most other OSS items are designed to run on Linux and many provide the functionality that users have come to see as essential, eg a graphical user interface.

Literally thousands of open source developments are under way, although these vary immensely in what they offer to typical end-users (as opposed to computer systems engineers, for example) and there is much duplication.

Office software, some open source, and other applications such as databases (eg from Oracle) are available to run on Linux. One particularly outstanding example of an open source application is the Apache web server software and some estimates give Apache and Linux respectively a 60% and 30% share of all web servers on the Internet. Products such as WINE allow Windows applications to run on Linux.

Despite all these initiatives, though, the current situation is that Linux has a far less rich set of applications available for it than has Windows. However, in the field of palmtops and similar devices, where the processing power is relatively limited and unit costs for the operating system are significant, Linux may easily gain the edge.

Linux and Obsolescence. Linux offers some significant advantages when combating obsolescence, but also raises some issues.

Advantages. As noted above, the operating system is a crucial element of the system when considering obsolescence. Because it is open source, Linux has some vital advantages over a commercial, closed source OS:

- Changes for bug fixes etc can be made in whatever way is appropriate, eg to retain the same interfaces for applications so that they do not need to change
- Hardware can be added or changed relatively easily since the drivers are readily accessible, and porting can be undertaken
- Changes to accommodate new requirements can be made as needed

Of course, these benefits arise simply because we have access to the source code, with all that gives in terms of visibility and control. In addition,

there are advantages that come from the "Linux culture":

- Others may have already developed and published a solution to the obsolescence problem
- If not, it may be possible to reduce cost and timescales by collaborating with others who share the problem
- A major element in combating obsolescence is having a plan for how future versions of the system will change to meet future needs, and there is great visibility of future Linux developments

Thus both the availability of the source and the broad development model of Linux offer many advantages. Naturally, though, there are some issues to be addressed.

Issues. If a customer, eg NATO or one of its member Ministries of Defence, decides using Linux in its systems is desirable, specifying this for a supplier is not trivial. Because of its modular nature and the wide variety of "distributions" available, a comprehensive list of modules, applications, etc is needed rather than a simple specification like "Windows NT version 4".

Much more significant is the question of how Linux might be exploited across a range of systems. Because of its portability and scalability, Linux lends itself to being used on many hardware architectures and there are obvious advantages in adopting it as a common platform. As well as countering obsolescence in the ways already discussed, this would also increase opportunities for re-use.

However, once we have a range of systems all using Linux, how do we manage the problem of upgrades, bug fixes, etc? A whole spectrum of options exist. We can simply highlight the problem and wait for the Linux community to solve it. This is directly analogous to the position with Microsoft and Windows. At the other extreme, we can actually make the change ourselves. We can also collaborate with other interested parties. In practice, it may be necessary to adopt a mixture of approaches, depending on individual circumstances.

A key question for defence systems with their typically long lives is: how long will the Linux community exist as it does now? Answering this requires predicting the future, of course, and so cannot be definitive, but the author's views are as follows.

Since the early 1990's the Linux community has grown rapidly to a size now of around 10 million. It is clear from looking at the various related web sites that there is a strong element of enthusiasm for the technical strengths of Linux, a major academic involvement, and a hint of religious wars against Microsoft and closed source software (not necessary all at the same time!). There are also echoes, for those old enough to remember, of the "Unix is about to rule the world" messages that have been appearing periodically over the last 25 years or so. Thus while the popularity of Linux is still growing, there is always the risk of something new catching the community's imagination as time passes.

Linux will never achieve the total market penetration of Windows, which despite its failings will remain the dominant operating system for desktops at least. Issues such as backwards compatibility and migration paths will be of less concern to the Linux community than to Microsoft (although the Microsoft route may not be easy and the "do it yourself" option is always available for Linux).

Thus while Linux will not disappear altogether, it seems very unlikely that in 20 years' time – perhaps even 10 – the community will exist as it does now. In many ways, Linux will then itself be obsolete! Given the cultural environment in which Linux has flourished, it may even be that visible exploitation by the defence community might hasten this shrinking of global support.

This does not, of course, mean that the defence world should dismiss Linux. What it does mean, though, is that if it is to adopt it widely, it must address this long-term issue.

A separate issue is related to the special defence needs of considering security, safety, etc. In principle, the answer is easy because all the Linux source is visible. In practice, although Linux is relatively simple compared with other operating systems, understanding adequately how it behaves is not easy.

More generally, even while a large Linux community exists, it can only be relied upon to provide some change if that modification has broad enough appeal. If the defence world is likely to need some more arcane work done (eg interfacing to a specialist device) – especially if it is urgent – then it is likely to be in the position of needing to be capable of doing it itself.

Finally, but by no means least important, is the question of verification and validation and confidence that the rather special way in which Linux is developed, released and controlled can provide a product that is suitably robust.

Again, the fact that the source is available, there is a large community of interest and Linux is relatively simple all help to suggest that at least if

a problem is found, it will be relatively easy to overcome. Neither, despite the impression its background might form in some minds, is it really obvious that Linux starts off any more likely to be faulty than a commercial OS. Nevertheless, this remains an issue to be addressed.

A Solution.

If the advantages of Linux are to be exploited, one way forward is to establish a "Linux Centre" for the domain of interest, eg the UK MOD or NATO. This Linux Centre could then act as the focus for Linux use across a number of systems in the customer's domain, see Figure 1.

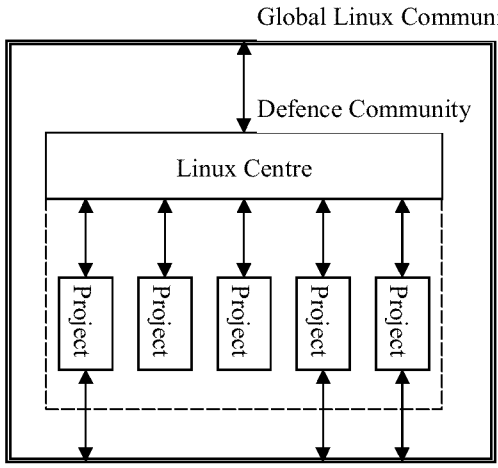

Figure 1 The Linux Centre

In particular the Linux Centre could:

- Be the repository for knowledge especially relevant to the domain (eg on security/safety issues)
- Offer the skill base necessary to make changes that for one reason or another cannot come from elsewhere
- Provide additional verification and validation functions to increase confidence in the Linux versions used
- Ensure commonality to whatever extent is appropriate, eg by defining the "standard" Linux distribution
- Synchronise upgrades, etc so that commonality is maintained
- Retain knowledge of older versions where appropriate
- Act as a clearinghouse for proposed changes to Linux: providing synchronisation across projects within the domain, supporting synergy between activities on separate projects, etc
- Act as an interface to the wider Linux community
- Retain knowledge, skills, etc as and when the larger community contracts

In essence, therefore, the Linux Centre would be a mirror within the specific defence community of the broader Internet foci that exist, such as Torvalds and various Internet sites. It would enable this broader community to be exploited, but reduce dependency on it.

Within the domain of defence systems, it would be important that the open source culture survived as far as possible, with projects interacting within themselves and more widely to produce rapid and collaborative solutions. However, the extra focus, synchronisation and longevity of the Linux Centre would ensure maximum benefit across the defence enterprise.

There are many options for the organisational form of the Linux Centre. Some of the aspects related to safety, for example, might be common with many non-defence domains, so sharing could take place. Also, support is commercially available now from a number of companies and it may be that some at least of these companies continue to offer support as the general community declines.

A Linux Centre is not without its own difficulties. As noted above, its mere existence may diminish the global support for Linux and thus undermine a key reason for using Linux in the first place!

Even an open source approach just within a limited defence community (eg on a national basis) raises issues of multi-project, multi-organisation working that would require a major re-think of some traditional attitudes in both procuring and supplying organisations.

There is also a question over balancing the open source culture of rapid "try it and see" refinement through collaboration with more traditional needs, practices and attitudes that tend to reflect increased levels of control.

However, widespread adoption of Linux without addressing these sort of issues entails a risk of being hit by what is essentially the obsolescence of Linux, with a much greater impact than if one had stayed with Microsoft!

The Open Source Future. Will the trend towards open source software in areas other than the operating system continue? Probably yes.

Operating systems are at the bottom of the OS/applications/service hierarchy of possible products vendor organisations might offer. Operating systems are no longer a major value

item for vendor or customer, and it is perhaps not surprising that the first real open source success is Linux. As the focus shifts more and more up the hierarchy, then the more likelihood there is of the levels that are "left behind" becoming open source as their value reduces:

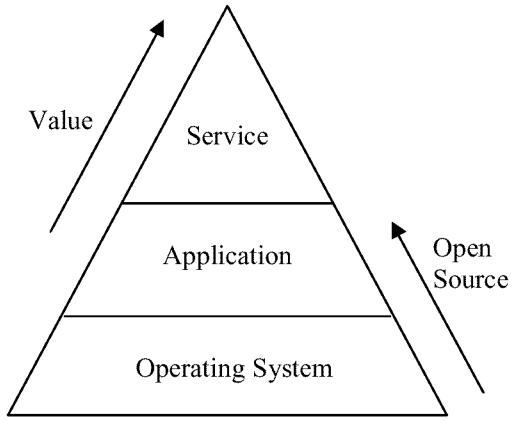

Figure 2 Open Source Evolution

On the other hand, the open source development model only has maximum value where there is a large community of interest who are enthusiastic to participate. As one climbs the hierarchy, there is an inevitable narrowing of interest as specialism increases. Another factor is the relative level of involvement of the academic community, which may be different for operating systems and word processors, say.

On balance, though, open source items seem likely to be a significant feature of the software world for some time.

Conclusion.

COTS items do offer major advantages in combating obsolescence, but their use does introduce other issues that must be addressed in a variety of ways.

The advantages are reduced if the COTS item is single-source and black box. In these circumstances, the vital aspects of visibility and control are severely weakened, if not lost altogether. Unfortunately, the operating system is one key system element that typically does have these undesirable attributes.

Linux, as an open source operating system with a very active user/developer community, offers much of the best of both worlds. It does not have to be developed from scratch, yet can be as visible and under control as a bespoke item. Also, it has already been ported to many hardware architectures, offering the possibility of a standard platform across a whole range of systems, with attendant opportunities for re-use, etc. Similarly, it offers the prospect of relatively easy porting to new architectures in the future.

However, Linux provides a much less rich set of applications than Windows and this seems likely to remain the case. In addition, the wide community of interest that has driven its success so far may well not be sustained over the long term. Even in the short term, the community addresses problems that are of interest and value to itself, and these may not coincide with defence system needs.

Thus while adoption of Linux is undoubtedly an attractive prospect in some ways, its widespread use introduces a number of issues. One way to address these is the establishment of a Linux Centre that can be a focus for defence needs and provide some degree of dedicated capability while still exploiting the broader community.

In addition, if the open source culture of rapid development and collaboration is to be retained, some change in approach from procurer and suppliers will be required.

More generally, the open source model is likely to spread, although its extent is difficult to predict. On balance, open source software provides a major opportunity to address at least some aspects of obsolescence. It should be exploited, but will require steps like those proposed for Linux to gain its full potential.